sevenstax

sevenstax Point-to-Point Protocol

PPP Specification and User Manual

Revision No.:	2.41
State:	Released
Author:	Peter Seliger, sevenstax GmbH

Initial version:	19/12/02
Last change:	03/03/03
Changed by:	Ralf Krizsan
Last Review:	21/02/03
Publication:	Public
Filename:	sevenstaxPPP_UserManual.sxw

Copyright (c) 2003 by sevenstax GmbH
This document is an intellectual property of sevenstax GmbH. Unauthorized copying and distribution is prohibited.

Table of Contents

1 Preface....2
2 How it works....3
3 Public Data Structures....7
4 Public Functions and Variables....9
5 Adjustable Parameters....11
6 External Services needed by sevenstaxPPP....13
7 Integration with TCP....14
8 Application defined Handler Procedure....14
9 Debugging Support....17
10 Currently Supported Add-Ons....18
11 Monitoring link quality....19
12 Example: Embedded System using sevenstaxPPP....20
13 Restrictions....22
14 List of Files....23
15 Change History....24

1 Preface

The Very Tiny Point-to-Point Protocol (sevenstaxPPP) is specially designed to connect embedded systems over a serial link (e.g. modem) to Internet Service Providers (ISP) and other hosts such as windows / Unix PC's. It is also designed to fit into small microcontrollers which have only a limited amount of memory. Therefore, code size and RAM usage are shrunken to a degree where PPP functionality just can be managed.

As a result of these restraints sevenstaxPPP is not a fully featured PPP implementation. In its basic settings sevenstaxPPP does not support any capability beyond PPP's fundamentals. To give an example ACCM, CHAP and header compression are disabled by default. There are some features that can be enabled depending on the user's requirements but the additional features will always result in larger code size. Please see chapter ' Currently Supported Add-Ons ' for details. Nevertheless even compiled with the default settings sevenstaxPPP is able to connect to ISPs, authenticate the user by PAP, receive an IP address with IPCP and deliver TCP packets to a local TCP/IP stack.

Other design goals of sevenstaxPPP were easy integration and maximum cooperation with existing software. To satisfy this needs, any call into sevenstaxPPP returns immediately and does not waste processor time while waiting for an event to occur. On completion, sevenstaxPPP calls an application-supplied handler procedure to notify events such as data reception, errors etc. Thus other software components will run smoothly and can do their tasks without being delayed.

The first version of sevenstaxPPP was build for microcontroller of Mitsubishi's M16C family. As sevenstaxPPP is entirely written in ANSI-C it can be adapted to other machines with minimum effort. Today sevenstaxPPP is also ported to some other microcontroller derivates like Infineon C167-family or NEC V850.

The hardware dependent settings are adjustable by changing the appropriate headers and recompiling the sources. Care has been taken trough the whole development process to an easy adaption of sevenstaxPPP to all specifics of common microcontrollers.

PLEASE NOTE: As a major difference to full featured PPP implementation like the PPP delivered with Linux or Microsoft Windows, sevenstaxPPP is not completely symmetric. Due to this and the restrictions mentioned above, sevenstaxPPP is intended to work in a client environment that connects passively (i.e. waits for the first LCP configure request).

2 How it works

The following is a description how sevenstaxPPP works internally.

- ***PPP framing***

  To send and receive packet-oriented data over serial lines single packets have to be distinguished and their integrity has to b ensured. On that score, PPP implementations are built atop a restricted subset of the HDLC (High Level Data Link Control) protocol. This protocol uses two special octet values, which are 7E and 7D in hexadecimal. The 7E is the frame delimiter and must be the last byte of any PPP frame. The 7D is the escape value meaning that every octet following a 7D must be exclusive-OR' ed with 20 (hexadecimal) to get the original value (on reception). PPP frames also contain a CRC value, the so-called Frame Check Sequence (FCS). This value is calculated before transmission and used by the receiver to verify packet data.

  There are some functions inside sevenstaxPPP regarding to this:

  - ***stxPPP__SendEscape()***

    Data is written directly to the serial link. stxPPP_SendEscape() looks for the presence of 0x7D, 0x7E or values smaller than 0x20. If such values are detected, stxPPP_SendEscape() replaces them by a 0x7D followed by the original value XOR'ed with 0x20. stxPPP_SendEscape() also updates the frame check value.

  - ***stxPPP_SendFrame()***

    This function takes a raw data packet and converts it to a valid PPP frame. The PPP frame will be sent out immediately using stxPPP_SendEscape().

  - ***stxPPP_Recv()***

    One of the most important functions is stxPPP_Recv(). It takes serial data byte by byte, removes PPP framing information, verifies the FCS and builds raw data packets.

- ***Protocol processing***

PPP frames contain sub-protocols. A list of those sub-protocols can be found at: http://www.iana.org/assignments/ppp-numbers.
After sevenstaxPPP's function stxPPP_Recv() has received a complete packet, it is delivered to stxPPP_DispatchPacket() which dispatches the packet according to its subprotocol number. If the sub-protocol is either LCP, IPCP or PAP, reply packets are build and sent back to the ISP. If it is a TCP packet, sevenstaxPPP calls the receive handler of TCP. Sub-protocols other then mentioned above are rejected which means a special LCP message is sent to the server telling him to stop the transmission of packets containing these protocols.

- ***Connecting to remote PPP implementations***

  After sevenstaxPPP is initialized, it is able to reply PPP frames sent by the counterpart. Generally ISPs do not send PPP frames without being stimulated. When stxPPP_Connect() is called, sevenstaxPPP first sends a short intentionally invalid PPP frame to the ISP to start the PPP session. This invalid frame shortens the delay an ISP needs to autodetect PPP mode. stxPPP_Connect() also instructs sevenstaxPPP send its own LCP (Link Control Protocol) configure request until the ISP responses. The configure request send by sevenstaxPPP contains no options (such as ACCM or MRU) which means that sevenstaxPPP only uses default parameters. Therefor the code can be kept small and simple and compatibility problems with different implementations are avoided as the default options have to be supported by every PPP implementation. Usually, after three or four repetitions, the first configure request from the ISP arrives.

  At this point, sevenstaxPPP must evaluate the ISP's PPP options and create proper replies.

  The ISP's options are handled as listed below:

  - ***Option 0 (Vendor extension)***

    This option will be rejected because it is has no meaning to sevenstaxPPP.

  - ***MRU***

    This option will be accepted because all PPP implementations are required to receive at least 1500 bytes and sevenstaxPPP does not create packets larger than this.

  - ***ACCM***

    As sevenstaxPPP has ACCM (Asynchronous control character map) support only as add-on, this option will be rejected by default. This means that sevenstaxPPP will use the PPP defaults and escapes all bytes from 0x00 through 0x1f. If ACCM support is included, then sevenstaxPPP accepts this option and subsequent data transmission will use ACCM.

  - ***Authentication Protocol***

    Most ISPs want to use CHAP as the authentication protocol. If CHAP is enabled (by #defining CHAP_SUPPORTED) sevenstaxPPP accepts CHAP as the authentication method. Otherwise, this option is negative acknowledged.

Instead of CHAP, sevenstaxPPP suggests using PAP which most ISPs will accept. There are some ISPs that do not accept PAP and send a terminate request as response which results in an abnormal termination of the dial-in.

- ***Magic number***

  Magic numbers are used to distinguish two peers and detect error conditions such as looped-back lines and inadvertently echoed data.
  This option will be accepted but the default version of sevenstaxPPP does neither store nor use the magic number. Magic numbers can be enabled by #defining MAGIC_SUPPORTED. In this case, sevenstaxPPP starts witch a constant value (PPP_START_ID) and tries to negotiate it. If the remote station declines this number, it is increased by one and send back with the next configure request. Magic numbers are useful for the autoping facility which will detect loopbacks more reliable by using the magic number.

- ***Protocol compression***

  This option is accepted if PPP_HEADER_COMPRESSION is #defined. In this case sevenstaxPPP also uses address compression. The result are slightly smaller PPP frames (two bytes less). If PPP_HEADER_COMPRESSION is not defined, protocol compression and header compression will be rejected.

- ***Address compression***

  All mentioned above (protocol compression) applies to this option without exception.

- ***Callback***

  This option is meaningless to sevenstaxPPP and will always be rejected.

- ***Any other option***

  Other options than mentioned above will be rejected.

Every single option will cause sevenstaxPPP to send an appropriate response to the remote peer. This will be done until sevenstaxPPP gets an LCP configure request which only contains the right options and parameters. Upon receiving and transmitting both LCP configure ACKs, sevenstaxPPP recognizes that the LCP option negotiation is over and it switches to the authentication process. Authentication with PAP is quite simple and therefore very suitable for sevenstaxPPP. A PPP frame containing password and user-name is built and transmitted repeatedly to the ISP until sevenstaxPPP receives an acknowledge. After receiving the acknowledge sevenstaxPPP enters IPCP which is the third stage of the connection process.
IPCP (Internet Protocol Control Protocol) is used to configure TCP/IP. IPCP uses a similar negotiation mechanism to LCP. This means that both stations have exchange IPCP configure requests until the other peer replies either an acknowledge or a reject. For sevenstaxPPP it is only relevant to receive a temporary IP address in

order to identify itself on the Internet. For this purpose, sevenstaxPPP's configure request only contains the option to get an IP address. All other options, an ISP might send, are rejected. First, sevenstaxPPP sends this option with an IP address of 0.0.0.0 telling the ISP that he should send the assigned IP address. When sevenstaxPPP receives the IP address, it is simply returned to the ISP which tells him that it has been accepted.

Now IPCP is configured and we are ready to run TCP/IP over PPP.

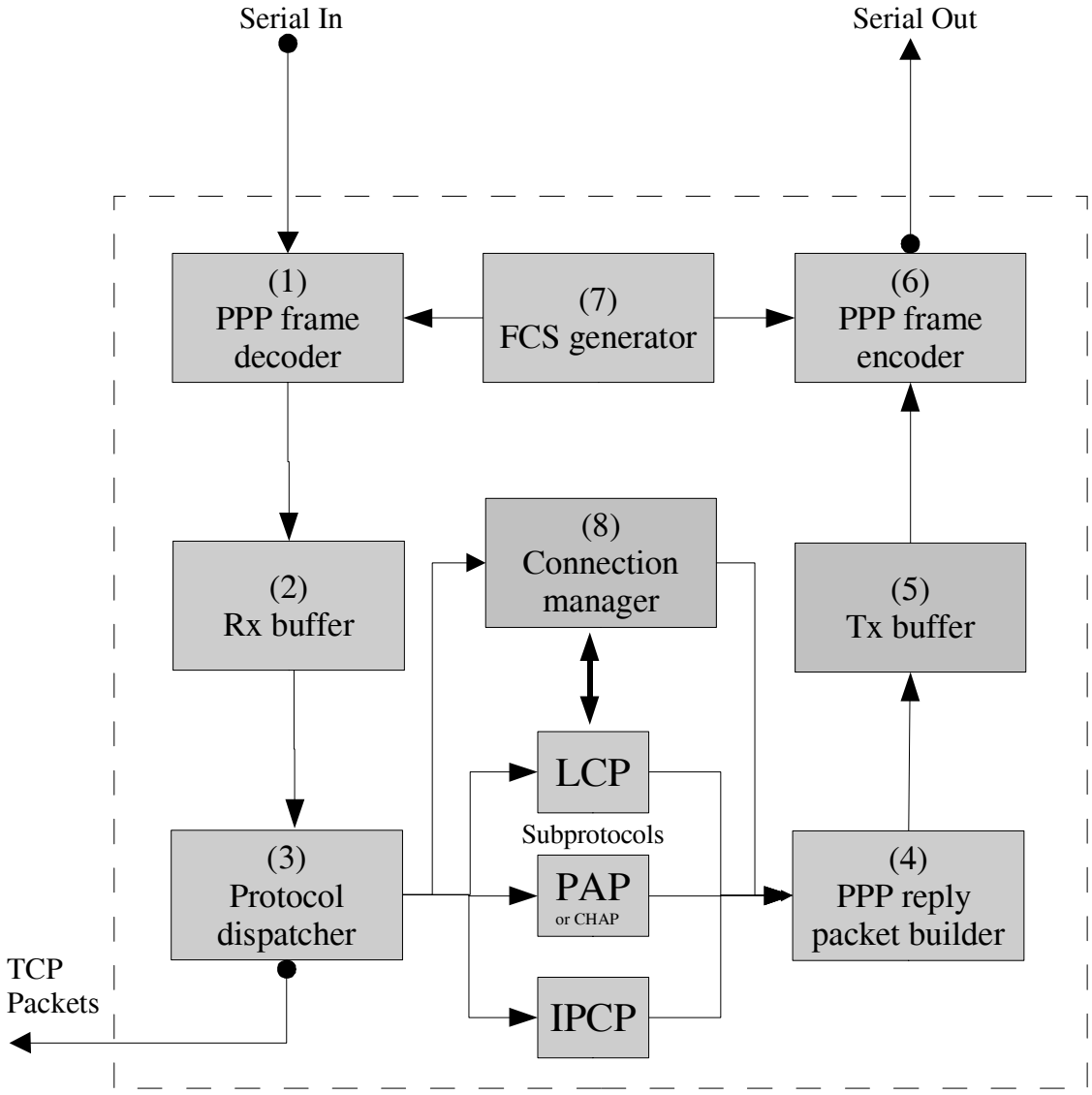

Fig 1. sevenstaxPPP block diagram

3 Public Data Structures

- ***The packet descriptor***

```
typedef struct _tag_PACKET_DESC
{
     PACKET_STATUS state;
     UINT16 size;
     UINT8 *data;
     UINT8 *payload;
} PACKET_DESC;
```

The packet descriptor is used to hold information about a data packet and for easy passing of this information between internal functions (by reference). The packet status member (PACKET_STATUS state) indicates whether the packet is in use (under construction) or ready for further processing. The member '*data' points to the location of the first byte in memory. The member 'size' specifies the length (in bytes) of the packet. The '*payload' member points to the first location after the PPP header. Packet descriptors are also used by sevenstaxPPP to deliver packets to higher network protocols (such as TCP).

Sometimes, only textual explanations may be hard to understand. Hence we show a drawing here which should remove contradictions on that.

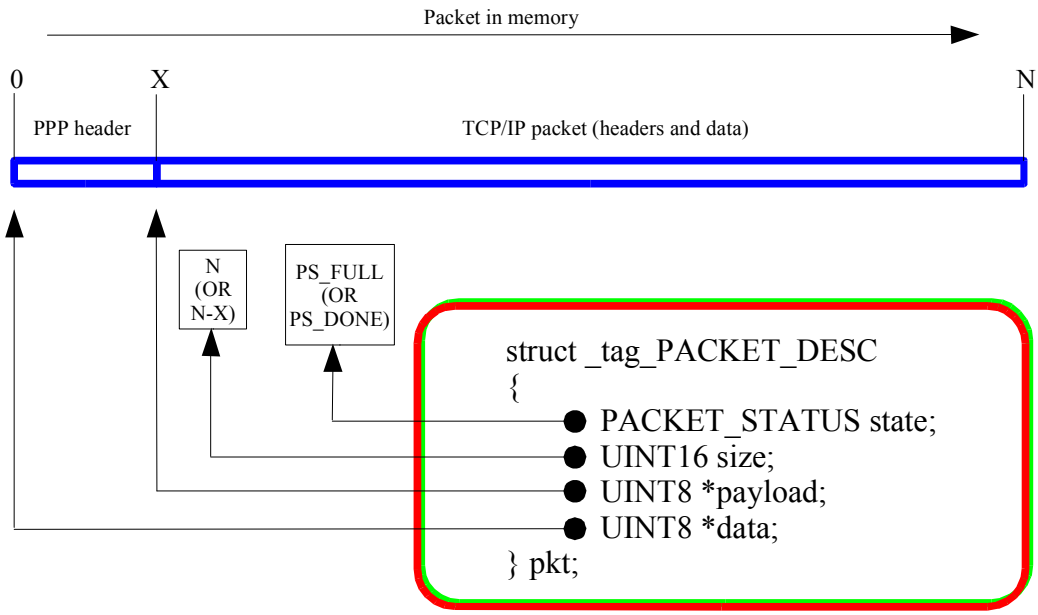

Fig 2. Packet descriptor

As you can see, the thin blue box represents a PPP packet containing an IP packet which resides in physical memory. The packet begins at offset 0, the IP packet starts at offset X and the whole PPP frame ends at offset N.

The red box is a Packet Descriptor which has been set up in order to reflect that packet.

As shown above, '*data' and '*payload' point to different locations where '*data' is always a pointer to offset 0 (which is the first address of the PPP header) and '*payload' always points to the beginning of data which is embedded into the PPP packet.

The 'size' member can be either the full length (N) or the length of the payload (N-X) depending on the use of the Packet Descriptor. When sevenstaxPPP calls its internal

functions, 'size' is always N which is the length of a whole PPP packet. When passing a TCP/IP packet (by a call to stxPPP_SendIPFrame()), 'size' is N-X which is the full length of the TCP/IP packet but without any PPP header information.

Because Packet Descriptors can either be owned by a producer or consumer, the member 'state' denotes the owner. A producer owns a Packet Descriptor when 'state' is PS_DONE. PS_DONE means that the Packet Descriptor is free and its buffer (*data) can be filled. When the producer has build a complete packet, it sets 'state' to PS_FULL which immediately transfers ownership to a consumer. The consumer now has access to the Packet Descriptor and can work on it. After the consumer has finished, it changes 'state' to PS_DONE which transfers control back again to the producer.

- ***The IP address***

```
typedef union
{
    struct
    {
            UINT8 b1;
            UINT8 b2;
            UINT8 b3;
            UINT8 b4;
    } ipb;
    UINT32 ipl;
} IPV4;
```

An IP address is actually a 32 bit value which is often shown as 4 decimal values, separated by dots, for better readability (e.g. 192.168.0.51). IP addresses are stored in network byte order where the first byte (b1) is the most significant byte.

- ***Link quality counters***

```
typedef struct _tag_PPP_STATISTICS
{
    UINT32 tx_frames;
    UINT32 rx_frames;
    UINT32 rx_errors;
    UINT32 tx_data;
    UINT32 rx_data;
    UINT32 negotiation_time;
} PPP_STATISTICS;
```

These structure contains some counters to log the PPP link quality. If PPP_STATS_SUPPORTED is #defined, sevenstaxPPP keeps a tally of all its traffic and updates an internal PPP_STATISTICS structure.

The structure members have the following meanings:

- **tx_frames**

  This is the number of PPP frames which are send to the lower layer until now.

- **rx_frames**

  This is the number of correct received frames (which have a valid FCS).

- **rx_errors**

  This is the number of bad frames (wrong FCS).

- **tx_data**

  This is the sum (in bytes) of all data which is send to the lower layer.

- **rx_data**

  This is the sum (in bytes) of data correctly received.

- **negotiation_time**

  The total time needed for LCP, PAP and IPCP negotiation.

4 Public Functions and Variables

sevenstaxPPP has only few interfaces. Some of them are functions and global variables which an application may use. Others are destined to higher level network protocols. See the following list for details:

- ***void stxPPP_Init (PROTOCOL_NOTIFY_HANDLER ppp_notify, SERIAL_READ_FUNC readfunc, SERIAL_WRITE_FUNC writefunc)***

  This function must first be called before one can use sevenstaxPPP. It needs three parameters. The first is a function pointer to a handler procedure which receives events from sevenstaxPPP. Whenever sevenstaxPPP has to report an event, it will call the handler procedure. The next two parameters are function pointers to the send and receive functions of a serial driver.

  stxPPP_Init returns immediately. Upon successful initialization, the handler procedure is called telling the application that sevenstaxPPP is ready to use.

- ***void stxPPP_Connect (STRING user, STRING pwd, UINT32 msecs_to_wait)***

  Calling this function will initiate the connection process. The user-name and the password must be supplied for PAP authentication. The third parameter is the time in milliseconds within which the PPP negotiations, including LCP, PAP and IPCP, must complete. If the negotiations are not finished after the time has elapsed, the handler procedure is called with an error code. Upon successful connection, sevenstaxPPP calls the handler procedure telling the application that TCP connections may be established.

- ***Void stxPPP_SendIpFrame (PACKET_DESC *pkt)***

  This function should be called by TCP/IP stacks. The packet descriptor's members '*data' and 'size' must have valid contents, '*payload' and 'state' are not used and should be zero. Before a TCP stack calls stxPPP_SendIpFrame() it should execute the following two steps:

  1. Build an entire TCP/IP packet starting at the 3rd byte of the send-buffer. This means that the first two bytes are omitted. So the 3rd byte of the send buffer contains the first byte of the IP header.
  2. Construct a packet descriptor and set the '*data' member to the beginning of your send buffer and the '*payload' member to the first byte of the IP header (or the

3rd byte of the send buffer). The 'size' member must be set to the number of bytes of the entire TCP/IP packet.

In other words: If your current TCP/IP packet has 500 bytes, it must be build at (or copied to) offset #2 of the send buffer. The '*data' member must point to offset #0, the '*payload' member must point to offset #2 and the 'size' member must contain the value '500'.

stxPPP_SendIPFrame() then increases the 'size' member by two, inserts the 'frame type' value for TCP/IP and calls stxPPP_SendRawFrame().

- ***Void stxPPP_SendRawFrame (PACKET_DESC *pkt)***

  This function is intended for the use by higher network protocols when they want to send data to the network using PPP. It does everything needed to build a PPP frame (HDLC coding, adding an FCS etc.) and sends it via the serial link.

  A packet descriptor (pkt) must be prepared properly before calling this function. The member '*data' must point to the first byte in memory and 'size' must be set to the number of bytes which should be send as PPP frame. The members '*payload' and 'state' are not used by stxPPP_SendRawFrame() and may contain any value.

- ***void stxPPP_Tick (void)***

  This is what keeps sevenstaxPPP alive. stxPPP_Tick() must be called as often as possible to ensure that sevenstaxPPP gets processing time and to conceive smooth and fast behaviour. Microcontroller environments without operating system usually have an endless loop from which all modules and functions are called. This loop is also a good place for calling stxPPP_Tick().

- ***void stxPPP_GetStats (PPP_STATISTICS *stats)***

  As an optional feature, sevenstaxPPP is able to record different counter values regarding to a PPP connection. If this is enabled (by conditional compilation through #define PPP_STATS_SUPPORTED), the application can read the counters by a calling stxPPP_GetStats(). The application must supply the address of a local PPP_STATISTICS structure which is filled by stxPPP_GetStats().

  Please Note: sevenstaxPPP's internal statistics counters are reset to zero when the PPP connection terminates.

  The following two variables are sevenstaxPPP's internal packet descriptors. These are pre-configured variables of type PACKET_DESC, which correlate to sevenstaxPPP's internal send and receive buffers.

- ***PACKET _DESC gPPP_recvpacket***

  Arriving data will first be decoded and then stored in memory. gPPP_recvpacket is the packet descriptor which points to received data. Higher level network protocols (e.g. TCP) have direct access to gPPP_recvpacket for the sake of fast and easy handling of received packets. This will also avoid expensive copy operations and save memory because sevenstaxPPP and TCP share the same region in memory.

- ***PACKET_DESC gPPP_sendpacket***

  A second packet descriptor is used to assemble packets for transmission. For the same reasons as gPPP_recvpacket, gPPP_sendpacket is also made public.

5 Adjustable Parameters

There are some parameters defined in 'pppdefs.h' which can be customized. These parameters should properly be adjusted depending on your hardware and communication device. The following list describes them in detail:

- ***PPP_RX_PKTSIZE*** *default: 600*

  This is the size of the receive buffer. The default value is 600 bytes. According to the RFCs any PPP implementation has to be able to receive packets of at least 1530 bytes. But, in fact, the value depends on the underlying protocol (TCP). If your TCP tells the other end that it is willing to receive e.g. 1280 bytes (always multiples of 256) per packet, then 1530 is a good choice combining the RFC conformity and the minimization of used memory. If your TCP only wants to get 256 bytes per packet then you might reduce RX_PKTSIZE to a smaller value.

- ***PPP_TX_PKTSIZE*** *default: 600*

  This is the size of the transmit buffer. The default value is 600 bytes. Internally, sevenstaxPPP does not need this much memory because it always deals with smaller packets. The reason why we need more memory is TCP. If your TCP transmits more than 512 bytes per packet, you should increase this value. Otherwise reduce it. It is not recommended that TX_PKTSIZE is smaller than 300 bytes (because TCP segments are a multiple of 256).

  Please note: The optimal packet size is depending on the application which is using PPP. To be fully RFC compatible this value has to be increased to 1530 bytes. Nevertheless the transportation layer (e.g. TCP) has also to be taken into this consideration. A small TCP packet size (e.g. 512 bytes) has advantages as well as disadvantages regarding the information flow-rate. On the one hand, if the connection is quite good and no retransmission is required larger packets will increase the information flow-rate. On the other hand retransmissions of larger packets will of course decrease the information flow-rate. For modem connections a PPP_RX_PKTSIZE of 600 bytes and a TCP packet size of 512 bytes is a good choice when using TCP/IP over PPP.

- ***MAX_PWDLEN*** *default: 32*

  Because of possibly retransmissions of PAP (or CHAP) requests, password and user name must be stored in memory. MAX_PWDLEN defines the size for both these arrays, the user-name and the password, used for authentication. The default value is 32 and suitable for the most ISPs. If you want to connect to an ISP who needs a larger password or user-name then you must increase MAX_PWDLEN.

- ***REQ_TIMEOUT*** *default: 3000*

  REQ_TIMEOUT defines the time interval in milliseconds for repeating own LCP requests. It means that the other peer should acknowledge our request within three seconds otherwise sevenstaxPPP will repeat it.

- ***PPP_PING_MAX_RETRIES*** *default: 6*

  When autoping is enabled PPP_PING_MAX_RETRIES defines the number of attempts to get a response from the remote system. After the 6th unsuccessful attempt sevenstaxPPP assumes the connection to be broken down and calls the application's handler procedure to report this event.

- ***PPP_PING_INTERVAL*** *default: 4000*

  This is the time interval in milliseconds after which sevenstaxPPP repeats an autoping. This parameter is only used when autoping is enabled.

- ***PPP_PING_THRESHOLD*** *default: 10000*

  sevenstaxPPP has a build-in mechanism to detect broken serial connections. This mechanism is called autoping and is enabled by defining AUTOPING_SUPPORTED in 'features.h'. The autoping is started when no data packet arrives within a certain time. sevenstaxPPP starts sending LCP echo requests to the remote peer and waits for responses. PPP_PING_THRESHOLD is the time in milliseconds within which the subsequent packets must arrive in order not to cause such echo requests.

- ***MY_ACCM*** *default: 0x00000000*

  ACCM means 'asynchronous control character map'. Because for some modems, characters below 0x21 are control values, so they must not appear in the data stream. PPP has a mechanism to transform them into two characters (escape sequences) which should not disturb modem operation. ACCM's are bitfields (32 bits) where every bit represents one of the first 32 ASCII codes. (0x00...0x20). The #define MY_ACCM is the ACCM value which will be used as initial negotiation value (if ACCM is supported). If MY_ACCM is set to zero (the default value) sevenstaxPPP announces not to need conversion of any ASCII code into escape sequences.

6 External Services needed by sevenstaxPPP

Like nearly all programs, sevenstaxPPP is depending on the assistance of other components. These additional components are described below:

The Serial Device Interface

To receive and transmit data over a serial device, sevenstaxPPP needs to call two functions (e.g. of a device driver) of the serial device driver. To connect sevenstaxPPP to such a

driver, just call stxPPP_Init() supplied with the addresses of the appropriate functions. Serial device functions suitable for sevenstaxPPP have to follow some specific requirements:

- ➔ The write function simply takes a single byte as a parameter and does not have a return value. Data should be transmitted as fast as possible and the function must return immediately. In case errors occur during writing to the serial port the current PPP frame will be corrupted and discarded by the remote peer.
- ➔ The read function takes a pointer to a single byte as a parameter and the return value depends on whether data is available or not. If there is something in the input buffer, the function must copy the byte to the supplied address and return zero. Otherwise, if the input buffer is empty, it simply returns a value different from zero.

If the driver does not export functions like this, wrapper functions have to be written to assure the functionalities mentioned above.

There are two entries in 'ppp.h' which can be used as function pointer definitions

```
typedef BOOL (*SERIAL_READ_FUNC)(UINT8 *)
typedef void (*SERIAL_WRITE_FUNC)(UINT 8)
```

The Counter Interface

In order to calculate timeouts etc., sevenstaxPPP needs a free running counter. This counter must output milliseconds as double words (32 bits). It is not required that the counter runs with an accuracy of 1ms, but the elapsed time between two arbitrary calls must be in milliseconds.

To gain access to such a system counter, sevenstaxPPP needs two entries defined in 'features.h'. The following example shows how one can adapt an OS supplied function (named GetTickCount()) to sevenstaxPPP.

```
#define GET_COUNTER_VALUE GetTickCount()
extern UINT32 GetTickCount (void);
```

7 Integration with TCP

As sevenstaxPPP is primarily designed to run TCP/IP over PPP, TCP must be able to send to and receive data from sevenstaxPPP.

For transmission, TCP should put a TCP/IP packet into sevenstaxPPP's tx buffer (gPPP_sendpacket), initialize a packet descriptor and call stxPPP_SendIpFrame() to transfer control to sevenstaxPPP. TCP can use a buffer of its own to build packets for transmission but ideally, it should use the one provided by sevenstaxPPP to avoid needlessly waste of memory.

On reception, sevenstaxPPP needs to call the TCP's receive function. For this purpose, 'tcpconnector.h' defines two entries to make it accessible to sevenstaxPPP. The following is an example where the function TCP_ProcessPacket() is made visible to sevenstaxPPP.

```
#define TCP_RECV(x) TCP_ProcessPacket(x)
```

```
extern void TCP_ProcessPacket (PACKET_DESC *pkt);
```

Every time, sevenstaxPPP recognizes a TCP packet, it calls TCP_ProcessPacket() through TCP_RECV. TCP receives a packet descriptor this way where the member '*payload' points to the first byte of the IP header. The member 'size' was set to the length of the entire PPP frame and '*data' points to the first byte of the PPP frame. Now it 's up to TCP. Generally, TCP does not need this information and should get all it needs from the TCP/IP headers where '*payload' points to.

PLEASE NOTE: Event though sevenstaxPPP can run alone, does not make much sense. PPP is a helper protocol for higher level network protocols such as TCP/IP or IPX/SPX.

8 Application defined Handler Procedure

To signal events, sevenstaxPPP needs a handler procedure provided by the application. Whenever sevenstaxPPP has something to report, it calls this handler procedure. By calling stxPPP_Init() sevenstaxPPP receives a function pointer which must point to the handler procedure. The definition of this function pointer will be found in 'notifycodes.h' and looks like this:

```
typedef void (*PROTOCOL_NOTIFY_HANDLER)(NOTIFY_CODE code, UINT32 lparam, UINT16 wparam)
```

The first parameter is called the NOTIFY_CODE. This is actually an identifier of the message which sevenstaxPPP has sent. The next two parameters may carry additional information depending on the message. NOTIFY_CODE is an enumeration type and is also defined in 'notifycodes.h':

```
typedef enum
{
    NC_PPP_INITIALIZED = 1,             // PPP is ready to use
    NC_PPP_CONNECTED,                   // PPP connection established
    NC_PPP_CONNTIMEOUT,                 // PPP connection timed out
    NC_PPP_SERVER_TERMINATED,           // PPP remote peer disconnected
    NC_PPP_PRIMARY_DNS                  // PPP has primary name server
    NC_PPP_SECONDARY_DNS                // PPP has 2nd name server
    NC_PPP_NEGOTIATION_STEP             // PPP LCP negotiation in progress
    NC_TCP_CONNECTED,                   // TCP connection established
    NC_TCP_CONNTIMEOUT,                 // TCP connection timed out
    NC_TCP_FIN,                         // TCP remote peer disconnected
    NC_TCP_RECV,                        // TCP data available
    NC_TCP_TXREADY                      // TCP ready to send
} NOTIFY_CODE;
```

With one of the first seven codes, sevenstaxPPP will call the handler procedure.
Below is a short description of their meanings. The last five codes of the enum refer to sevenstaxTCP and are not used by sevenstaxPPP and thus are not described here.

- ***NC_PPP_INITIALIZED***

  This event occurs after the application has called stxPPP_Init(). When receiving NC_PPP_INITIALIZED, the application can be sure that sevenstaxPPP has finished initialization and is ready to work. The additional parameters are not used.

- ***NC_PPP_CONNECTED***

  Result code if stxPPP_Connect() terminates successfully. When all sub-protocols, LCP, PAP and IPCP have completed, sevenstaxPPP will signalize success with this event. The first parameter contains the IP-address which the ISP has assigned to us (MSB first). The second parameter is not used.

- ***NC_PPP_CONNTIMEOUT***

  Result code if stxPPP_Connect() terminates unsuccessfully. In case that either LCP, PAP or IPCP do not complete within the time specified by stxPPP_Connect(), sevenstaxPPP generates this event. Both additional parameters are not used.

- ***NC_PPP_SERVER_TERMINATED***

  This event occurs either when the remote peer wants to terminate a connection or when autoping is enabled and sevenstaxPPP has detected that the counterpart does not respond anymore. NC_PPP_SERVER_TERMINATED is generated every time sevenstaxPPP receives a 'terminate request' either from LCP or IPCP. Terminate requests can appear at any time. During the connection process LCP terminate requests may be received if the server detects an error. It is important to know that such an event might be an evidence to serious errors and should be handled by the application. The first parameter indicates which subprotocol causes the event. If it is 0xc021 then it was an LCP terminate request. If it is 0x8021 then the terminate request came from IPCP. If it is 0x0000, autoping has produced this event. The second parameter is not used.

- ***NC_PPP_PRIMARY_DNS***

  This event only occurs, if DNS support is enabled. The DNS support can be switched on at compile time by #defining DNS_SUPPORTED in 'features.h'.
  In this case, if the ISP advertises a name server's IP address, sevenstaxPPP accepts and forwards the address to the application using the first parameter (lparam) of the handler procedure. The most significant byte of lparam contains the first number of the IP address. The second parameter (wparam) is not used and always zero.

- ***NC_PPP_SECONDARY_DNS***

  It is possible that a name server is temporarily not reachable for various reasons. As access to the domain name system is essential for many applications, most ISPs send a second name server's IP address which can be used when the first one is down. All mentioned at NC_PPP_PRIMARY_DNS also applies to NC_PPP_SECONDARY_DNS.

- ***NC_PPP_NEGOTIATION_STEP***

  Before sevenstaxPPP posts an NC_PPP_CONNECTED event to the application, two times an NC_PPP_NEGOTIATION_STEP event is issued to notify the application of the initial negotiation progress. The long parameter (lparam) contains an identifier to show which protocol has completed negotiation. It can have one of two possible values:

- 0: LCP negotiation has completed successfully
- 1: Authenticated with PAP or CHAP

When the application (successfully) calls stxPPP_Connect() it receives events in the following order:

- NC_PPP_NEGOTIATION_STEP, lparam is 0
- NC_PPP_NEGOTIATION_STEP, lparam is 1
- NC_PPP_PRIMARY_DNS (optional)
- NC_PPP_SECONDARY_DNS (optional)
- NC_PPP_CONNECTED

There are three general rules for executing code inside the handler procedure:

➔ **Leave as fast as possible**

When sevenstaxPPP signals events, it cannot proceed before the handler procedure has exited. If you do lengthy tasks inside the handler procedure, the receive buffers may overrun, packets will get lost and will have to be repeated. This will cause appreciable delays and might affect performance of the whole system. So be sure not to waste much time here.

➔ **Never call sevenstaxPPP directly**

Because the handler procedure is called from inside sevenstaxPPP, it is dangerous calling back into sevenstaxPPP. Doing so can cause deadlocks or even unlimited recursions which will result in stack overflows and thus crash the system.

➔ **Do not write to RX/TX buffers**

sevenstaxPPP might be in a state where it waits for some data or builds a packet for transmission. Therefore, changing buffer contents can cause sevenstaxPPP to misbehave.

An example of how such a handler procedure may look like is given below:

```
void  PPPHandlerProc (NOTIFY_CODE n, UINT32 lparam, UINT16 wparam)
{
     switch (n)
     {
            case NC_PPP_INITIALIZED:
            // Tell the application that sevenstaxPPP is ready to run
            break;

            case NC_PPP_CONNECTED:
            // We could now run TCP/IP
            // to download websites etc.
            break;

            case NC_PPP_CONNTIMEOUT:
```

```
                // We might retry or select another ISP
                break;

                case NC_PPP_SERVER_TERMINATED:
                // The ISP is about to hang up
                // We should retry or finish here
                break;
        }
    }
```

9 Debugging Support

At the source level, sevenstaxPPP contains a simple debugging support through a macro called DPRINT(). Like printf(), DPRINT() has a variable length argument list. To switch on debug outputs you must compile sevenstaxPPP with DEBUG defined and customize the implementation in debug.h which is platform specific.

PLEASE NOTE: If you want to use DPRINT() yourself and it follows an if/else statement etc., be sure to enclose it in braces to avoid different behaviour of the debug and the release version of the code.

10 Currently Supported Add-Ons

sevenstaxPPP can be extended through various components. In 'features.h'

- ***Automatic link verification (#define AUTOPING_SUPPORTED)***

  Since there is no universal way to detect if the other station (modem) is still available, sevenstaxPPP has the feature to permanently check for the existence of the remote peer. If AUTOPING_SUPPORTED is define when the PPP sources are compiled, this feature will be activated. If no data was received for a certain time, sevenstaxPPP repeatedly sends LCP echo requests to the remote peer. In case the consecutive echo requests do not cause a reply, the remote peer is assumed to be unavailable and sevenstaxPPP notifies the application through the handler procedure (NC_PPP_SERVER_TERMINATED).

  To enable autoping, AUTOPING_SUPPORTED must be defined before compilation and 'features.h'. Some defines in ' pppdefs.h ' should be adjusted for a proper operating of autoping:

  - ***PPP_PING_MAX_RETRIES*** *default: 6*

    This is the number of pings that may fail, before the remote station is assumed to be unavailable.

  - ***PPP_PING_INTERVAL*** *default: 4000*

    This is the time in milliseconds, the pings are repeated.

  - ***PPP_PING_THRESHOLD*** *default: 6000*

Within this time (in milliseconds) packets must arrive. Otherwise pings are issued

- ***Magic number support (#define MAGIC_SUPPORTED)***

  It is possible that autoping malfunctions, when the other station becomes unavailable and the local modem changes from data mode into command mode. In command mode all data may be echoed and sevenstaxPPP answers echo requests by itself. To prevent such misbehaviour, sevenstaxPPP can exchange so-called magic numbers with the remote station. A magic number is a unique identifier, which will be send as part of some packets such as echo requests and replies. A PPP implementation, which is aware of both magic numbers, the own and the peer's, is not susceptible to errors caused by loopbacks.

  To enable magic number support, one must #define MAGIC_SUPPORTED. No additional source codes are needed. sevenstaxPPP uses a constant value 'PPP_START_ID' from 'pppdefs.h' as its initial magic number. This value is currently defined as 0x305ab6c1 but can be changed if desired.

- ***ACCM support (#define ACCM_SUPPORTED)***

  Some modems interpret values below 32 decimal not as data bytes but as control characters. Therefor PPP implementations replace these characters with escape sequences. For modems which are not affected by such bytes (nevertheless they use XON/XOFF characters), escaping means unnecessary overhead and reduces communication speed. To avoid this problem PPP uses a value, which is a bitmap of 32 bits (the ACCM) where a 'one' represents a character that must be escaped and a 'zero' stands for characters that can be send without any change. ACCM values are exchanged during the LCP negotiation process.

  To enable ACCM support, ACCM_SUPPORTED (also see 'features.h') must be defined before compilation. If ACCM support is enabled, upon reception of the ACCM from the other end, sevenstaxPPP will combine its own ACCM (MY_ACCM defined in 'pppdefs.h') with the ACCM from the remote peer using the logical 'OR' operator. The resulting ACCM will then be used in escaping bytes sent by sevenstaxPPP. If the peer does not send an ACCM, sevenstaxPPP will use MY_ACCM only.

- ***Challenge authentication protocol (#define CHAP_SUPPORTED)***

  sevenstaxPPP has a build-in optional authentication scheme called 'CHAP'. In cases, where a remote PPP does not accept PAP as authentication protocol, CHAP can be used for a safer kind of authentication.

  To enable CHAP you must do the following:

  - In 'features.h' insert or uncomment '#define CHAP_SUPPORTED'
  - include the files 'md5.c' and 'md5.h' into your project.

  If CHAP is enabled, sevenstaxPPP first tries to negotiate CHAP as its default authentication protocol. Anyhow, if CHAP is not accepted by the remote peer,

sevenstaxPPP reverts to PAP. Authentication fails if the remote PPP neither knows CHAP nor PAP.

11 Monitoring link quality

While sevenstaxPPP is running it is possible for the application to get some counter values regarding the link quality. Link quality monitoring must be enabled at compile time by '#define PPP_STATS_SUPPORTED from 'features.h'.

During connected state, this means when both PPP's have negotiated successfully, the application can query link quality counters by calling stxPPP_GetStats() supplied with the address of its private PPP_STATISTICS structure. Every time the application calls stxPPP_GetStats(), it receives fresh copies of the current counter values.

All counters are reset to zero, when sevenstaxPPP starts a new negotiation process.

For a description of PPP_STATISTICS please see the chapter 'Public Data Structures'.

12 Example: Embedded System using sevenstaxPPP

The following is an example of how sevenstaxPPP can be integrated into a simple embedded system. Please note that this is not a functional source code rather an example to make things more obvious.

```
// First we have to include some PPP related stuff
#include "features.h"      // for platform dependent definitions
#include "ppp.h"   // for prototypes
#include "pppdefs.h"       // for typedefs etc.

// This will get TRUE if sevenstaxPPP cannot connect
BOOL ppp_connection_error ;

// The handler procedure which receives
// events from sevenstaxPPP
void  PPPHandlerProc (NOTIFY_CODE n, UINT32 lparam, UINT16 wparam)
{
     switch (n)
     {
             case NC_PPP_INITIALIZED:
             // E.g. Initialize TCP after PPP
             TCP_Init();
             break;

             case NC_PPP_CONNECTED:
             // Connect to a web server
             TCP_Connect (IP_ADDRESS, PORT, TIMEOUT);
             break;

             case NC_PPP_CONNTIMEOUT:
             // Set the errorflag if sevenstaxPPP cant connect
             ppp_connection_error = TRUE;
             break;

             case NC_PPP_SERVER_TERMINATED:
             // The remote peer wants to terminate
             // We close the TCP connection
             TCP_Disconnect();
             break;
     }
}
```

```
// Entry point of the program
void main (void)
{
     // We now dial-in to an ISP
     ModemDial (ISP_NUMBER);

     // Next we run sevenstaxPPP and connect
     stxPPP_Init (PPPHandlerProc, TRUE);
     ppp_connection_error = FALSE;
     stxPPP_Connect ("user", „password", 60000);

     // This is the endless loop from which
     // all services will be called
     while (1)
     {
            // sevenstaxPPP  gets processor time here
            stxPPP_Tick();
            // Try to recover from connection errors
            if (ppp_error_flag == TRUE)
            {
                   ppp_error_flag = FALSE
                   // Re-Initialize
                   TCP_Disconnect();
                   stxPPP_Init (PPPHandlerProc);
                   stxPPP_Connect („user", „password", 60000);
            }
            // Call other system services
            ...
            ...
            ...
     }
}
```

13 Restrictions

sevenstaxPPP is focused on systems with very small resources, to give them relevant PPP functionality. Therefore – in comparison with a full featured PPP implementation of a PC software f.e. - it has some restrictions in usage and capabilities, the user should know:

#	*Restriction*	*Details*
1	Not aware of interrupts	If parts of sevenstax embedded protocols are called from within an ISR, synchronization must be done by the user.
2	Not full RFC compliant	To give maximum capabilities within small system resources, sevenstax embedded protocols fulfil only basic and most relevant parts of the RFC specifications.
3	Server-side operation	The current sevenstaxPPP version works well on clients but lacks some features suitable for server applications. LCP negotiation is fully developed, but only client parts of IPCP are there. This means that sevenstaxPPP doesn't offer IP addresses and name servers. This will be point of improvement in next release.
4	Polling	On every call to stxPPP_Tick(), sevenstaxPPP polls a byte from a serial I/O port or queue.
5	PPP frame size	In order to get along with as many ISPs as possible, sevenstaxPPP does not send MRU values and accepts every MRU during LCP negotiation. The real MRU of sevenstaxPPP is PPP_RX_PKTSIZE. If sevenstaxPPP's receive buffer is configured large enough e.g. 600 bytes, a TCP which needs two segments (512 bytes), will work well. Frames which don't fit are rejected.

sevenstaxTCP is object of permanent improvement. Please contact info@sevenstax.de to get more information about the newest PPP features given in next release version.

14 List of Files

The following is a list of files (source codes) which are needed to integrate sevenstaxPPP into your project.

File	*Description*	*sevenstax PPP*	*sevenstax optional*
ppp.c	PPP protocol driver source code	x	
ppp.h	Application callable functions (prototypes) of sevenstaxPPP	x	
pppdefs.h	Constants and definitions used by sevenstaxPPP	x	
pktdesc.h	Packet analyse types and macros	x	
features.h	Enable/disable sevenstaxPPP features at compile time	x	
notifycodes.h	Handler procedure definition and NC_xxx codes	x	
wordaccess.h	Macros for big/little endian conversion and bytewise access on aligned processors	x	
ipv4.h	IP address structure, associated macros and definitions	x	
standard_types.h	Target system basic types	x	
tcpconnector.h	sevenstaxPPP upper API function prototypes to be used with f.e. with sevenstaxTCP	x	
runtimelib.h	Connectors to c-library functions.	x	
runtimelib.c	Replacements for c-library functions.		x
md5.c	CHAP authentification sources		x
md5.h	CHAP authentification interface		x
debug.h	Debug functions prototypes		x
winstart.c	Demo source, can be used to force a Windows DialUp-Server to begin PPP negotiations		(x)
modem.h	Demo Source, modem driver code, header file if used for PPP		(x)
modem.c	Demo Source, modem driver demonstration code, simulation under Windows NT / 2000, uses PCs COM-Port		(x)
main_ppp.c	Demo Source, PPP main() demo code used in conjunction with sevenstaxTCP, sending HMTL string when PPP+TCP connected		(x)

(x) These files are only needed in the test environment

15 Change History

Vers.	*Date*	*by*	*Change description*
1.42	20/08/02	pe	Initial version
2.20	20/01/03	pe	Additional features
2.3	12/02/03	krz	some corrections
2.4	21/02/03	krz	Reviewed
2.41	26/02/03	pe	restrictions added

Copyright (c) 2003 by sevenstax GmbH

This document is an intellectual property of sevenstax GmbH. Unauthorized copying and distribution is prohibited.